REAL-TIME CALCULUS FOR SCHEDULING HARD REAL-TIME SYSTEMS

Lothar Thiele, Samarjit Chakraborty and Martin Naedele

Swiss Federal Institute of Technology Zurich, Computer Engineering and Networks Lab.
CH-8092 Zurich, thiele@tik.ee.ethz.ch

ABSTRACT

This paper establishes a link between three areas, namely Max-Plus Linear System Theory as used for dealing with certain classes of discrete event systems, Network Calculus for establishing time bounds in communication networks, and real-time scheduling. In particular, it is shown that important results from scheduling theory can be easily derived and unified using Max-Plus Algebra. Based on the proposed network theory for real-time systems, the first polynomial algorithm for the feasibility analysis and optimal priority assignment for a general task model is derived.

1. INTRODUCTION

As an application of the real-time calculus proposed in this paper we will consider preemptive task scheduling with static priorities. In this case, the scheduler evaluates the set of tasks ready to run after each relevant event. This may result in the currently executing task being stopped - preempted - and another task to be run instead. The scheduler uses priorities to determine the next job to be scheduled. These are calculated at design time and never change during execution. Main problems are the priority testing (given a system of tasks and associated priorities, will all tasks complete by their deadlines) and priority assignment (given a system of tasks, determine priorities such that the tasks will complete by their deadlines). The standard techniques for priority assignment are rate monotonic scheduling (RMS) and deadline monotonic scheduling (DMS) and their extensions. See [1] for an overview.

Recently, an algebraic approach for constructing mathematical models of discrete event systems has become popular. One of the intention is to describe temporal properties of certain classes of queuing systems, see [2]. Detailed discussions about the underlying algebra (max-plus) and their historical roots can be found in [3]. The rich mathematical structure allowed to obtain strong results in a variety of application areas, see e.g. [2, 4] and the references therein. Recently, these results have been applied to the analysis and design of packet networks like ATM, see [5].

Network Calculus [6, 7, 5, 8] is a mathematical approach to model network behavior. In a unified representation, it allows for computing tight bounds on delay and backlogs in guaranteed service protocols like the IETF framework, for understanding and calculating quality of service parameters in differentiated services, for computing the influence of traffic conditioning, for dealing with packet scheduling algorithms like guaranteed rate and weighted fair queuing, and much more, see e.g. [5, 8].

In this paper, the link between the linear system theory of discrete event systems and communication networks is extended to real-time systems. In particular, it is shown how to model task systems and their dependencies, computing resources and scheduling algorithms, among them being the multiframe task model [9], the recurring real-time task model [10], the bursty task model [11] and the variable task model [12]. For static priority scheduling, we obtain algorithms for feasibility analysis and optimal priority assignment which generalize and unify previous results, for example the rate-monotonic analysis [13, 14]. Based on the new network theory for real-time systems, the first polynomial algorithm for a general class of task models is described.

2. MODELS

2.1. Basic Model

The basic network model of a processing resource in the presence of incoming task requests is shown in Fig. 1 on the left. It characterizes a processing resource that receives incoming requests and executes them using the available capacity. To this end, some non-decreasing functions are introduced.

The *request function* $R(t) \geq 0$ represents the total amount of computation that has been requested up to time t. The *delivered computation function* $R'(t) \geq 0$ represents the total amount of computation that has been delivered up to time t. The *capacity function* $C(t) \geq 0$ represents the maximum amount of computation that could be delivered up to time t (if the processor runs under full load). The *remaining capacity function* $C'(t) \geq 0$ represents the total amount of computation that has not been used up to time t.

Using the above definitions, the relationship between the four quantities can be stated, see Prop. 1 and Fig. 1.

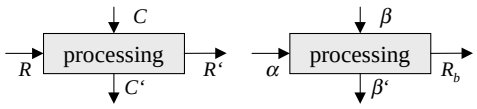

Figure 1: Models of processing task requests on a computation resource.

Proposition 1 (Basic Model) *For a processing node characterized by the capacity function $C(t)$ and the incoming requests function $R(t)$ we have $C'(t) = C(t) - R(t)$ and*

$$R'(t) = \min_{0 \le u \le t} \{R(u) + C(t) - C(u)\}$$

The following example explains some of the above definitions.

Example 1 *Let us suppose that the given processor service capacity and task requests can be described as in Fig. 2. The request function R may correspond to a situation where at $t = 1$ and $t = 3$ there are incoming tasks which request a computation demand of 2 and 1, respectively. For $t \geq 4$, there is a continuous demand of 1/2 per unit time. The capacity function C may model the situation of a processor delivering 1 computation unit per time unit for $0 \leq t \leq 2$ and $t \geq 5$. Because of a cycle stealing, it is able to deliver only half a unit for $2 \leq t \leq 4$ and stops completely for $4 \leq t \leq 5$ because of a high priority interrupt.*

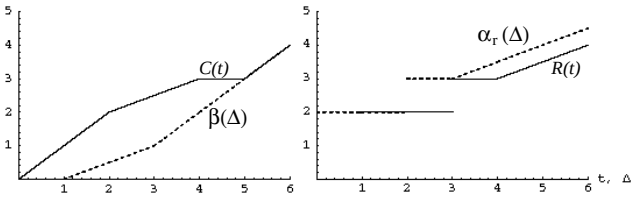

Figure 2: Example of a request function R, its request curve α_r, a capacity function C and its delivery curve β.

2.2. Bounds

Usually, the task requests or the behavior of a processor are not known at compile time. In order to deal with hard real-time bounds, we are interested in characterizing classes of computation requests and processor capacities.

[11], [9] and [10] characterize tasks by the maximum amount of computation requested within any interval of a certain size. This approach can be formalized by adapting the concept of arrival curves from the networking domain for task releases.

Definition 1 (Request Curve) *The request curve α_r of a request function R is a non-decreasing non-negative function which satisfies $R(t) - R(s) \leq \alpha_r(t - s) \quad \forall s \leq t$.*

For all practical situations only sub-additive request curves ($\alpha_r(s + t) \leq \alpha_r(s) + \alpha_r(t)$) make sense. Using a closure operation in max-plus algebra one can construct the maximum sub-additive function smaller than α_r. Given a request function R the smallest (sub-additive) request curve α_r can be constructed as shown below.

Proposition 2 (Request Curve) *For a given request function R, the minimum request curve α_r can be calculated as*

$$\alpha_r(\Delta) = \max_{u \geq 0} \{R(\Delta + u) - R(u)\}$$

Usually processors serving task requests are considered to be able to deliver a constant amount of computation per unit time. This is, e.g., the implicit basic assumption for the well-known results of [14] as well as the feasibility tests of [11], [13] and [10]. The more general concept of a delivery curve, which is inspired by the use of service curves in the networking domain, characterizes a processor via the minimum amount of computation that is guaranteed to be delivered in an interval of length Δ starting at any arbitrary point in time t.

Definition 2 (Delivery Curve) *The delivery curve β of a capacity function C is a non-decreasing non-negative function which satisfies $C(t) - C(s) \geq \beta(t - s) \quad \forall s \leq t$.*

Again, mainly a super-additive request curve ($\beta(s+t) \geq \beta(s) + \beta(t)$) make sense. Using a closure operation in max-plus algebra one can construct the minimal super-additive function larger than β.

Proposition 3 (Delivery Curve) *For a given capacity function C, the maximum delivery curve β can be calculated as*

$$\beta(\Delta) = \min_{u \geq 0} \{C(\Delta + u) - C(u)\}$$

2.3. Processing of Bounds

Now, we are interested in the transformation of the request curve and delivery curve, see Fig. 1 right hand side. In particular, we are interested in the remaining delivery curve $\beta'(\Delta)$ which bounds the remaining capacity function $C'(t)$ according to Prop. 1. The proof of the following proposition can be found in [15].

Proposition 4 (Remaining Delivery Curve) *Given request and capacity functions R and C, bounded by the request and delivery curves α_r and β respectively, C' according to Prop. 1 is then bounded by the delivery curve*

$$\beta'(\Delta) = \max_{0 \le u \le \Delta} \{\beta(u) - \alpha_r(u)\}$$

In a similar way, we can give the following bounds on the delivered computation function R':

Proposition 5 (Delivered Computation Bounds) *Given request and capacity functions R and C, bounded by the request and delivery curves α_r and β respectively, R' according to Prop. 1 is then bounded by the request curve*

$$\alpha'(\Delta) = \max_{u \geq 0}\{\alpha_r(\Delta + u) - \beta(u)\}$$

and by the expression

$$R'(t) \geq R'_b(t) = \min_{0 \leq u \leq \Delta}\{R(u) + \beta(t - u)\}$$

It has been shown in [15], that these bounds are tight.

Example 2 *Fig. 3 shows the remaining capacity C' and the delivered computation function R' (see Prop. 1), their bounds β' (see Prop. 4) and R'_b (see Prop. 5). The corresponding request function R, request curve α_r, capacity function C and delivery curve β are shown in Fig. 2.*

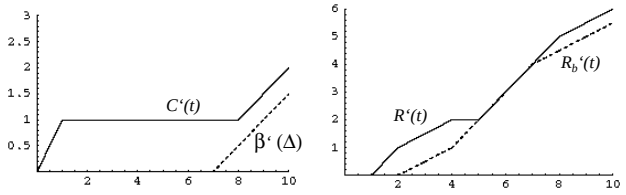

Figure 3: Functions and bounds after processing.

2.4. Calculation of Bounds

Before dealing with the use of the real-time calculus for fixed priority scheduling, we will show how the request curve for a realistic task system can be computed in polynomial time. Previous approaches suffered from exponential complexity, see [16, 10].

The task model is similar to that described in [16] where a task T is represented by a task graph $G(T)$ which is a DAG with a unique source and a sink vertex, see Fig. 4. Each vertex u of the graph $G(T)$ represents a subtask, and associated with it is a pair $(e(u), d(u))$ which means that it requires $e(u)$ units of time to execute and has to be completed within $d(u)$ units of time after it is triggered. Each directed edge (u, v) of $G(T)$ represents a control flow with conditional branches and associated with it is a parameter $p(u, v)$ denoting the minimum amount of time that must elapse before vertex v can be triggered after the triggering of vertex u. If the vertex triggered is the sink vertex of $G(T)$, then the next vertex to be triggered is the source vertex and it can be triggered at any time at or after $P(T)$ units of time has elapsed since its last triggering. $P(T)$ is called the *period* of the task T.

As defined above, the *request curve* $\alpha_r(\Delta)$ denotes the maximum computation demand in an interval Δ, i.e. the maximum sum of computation times $e(u)$ of subtasks having their triggering times within an interval of size Δ. For the scheduling test and priority assignment we will also need the *demand curve* $\alpha_d(\Delta)$. It is the maximum sum of computation times $e(u)$ of subtasks having their triggering times and deadlines within an interval of size Δ.

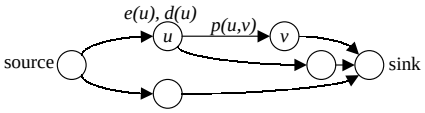

Figure 4: Example of a task graph with conditional branches.

The algorithm presented in this section computes the values of $\alpha_r(\Delta)$ and $\alpha_d(\Delta)$ for event sequences in which the source vertex is triggered at most once. Other cases can be reduced to these sequences, see [16]. These event sequences can be represented by paths in the following graph: two copies of the task graph $G(T)$ are joined by joining the sink node of one copy with the source node of the other copy. To compute $\alpha_r(\Delta)$ and $\alpha_d(\Delta)$ for such event sequences, [16] enumerates all the possible paths in this graph (exponential complexity). We show how to compute $\alpha_d(\Delta)$ in polynomial time. $\alpha_r(\Delta)$ can be computed with the same algorithm by assuming $d(u) = 0$.

Since the graph resulting out of joining two copies of the task graph is a DAG, we can assume that the vertices of this graph are labeled with integers and there can be a directed edge from a vertex i to a vertex j only if $i < j$. The algorithm is incremental. It processes the nodes of the graph in an increasing order of their labeling, updating two sets of tuples at each step. Let L^i be a set of tuples (f, Δ) describing all sequences of executions of subtasks with vertex i as the last subtask. In (f, Δ), f is a lower bound in the execution demand in interval Δ. Let $L(i)$ be a list of tuples corresponding to all possible execution sequences involving vertices only up to and including i. Assuming $L(i)$ and L^i are computed for all $i = 1, \ldots, j$, one iteration of the algorithm to compute $L(j+1)$ and L^i is the following:

$L^{j+1} = \{(e(j+1), d(j+1))\}$
for each edge $(k, j+1)$ **do**
  for each tuple $(f, \Delta) \in L^k$ **do**
    $L^{j+1} = L^{j+1} \cup$
      $\{(f + e(j+1), \Delta + p(k, j+1) + d(j+1)$
      $-d(k))\}$
$L(j+1) = L^{j+1} \cup L(j)$
Reduce the sets L^{j+1} and $L(j+1)$

The algorithm described above uses the following procedure for reducing a set of tuples $(f_1, \Delta_1), \ldots, (f_k, \Delta_k)$. It removes all tuples representing less restrictive bounds on the execution demand than those expressed by other tuples already contained in the set.

For each pair of tuples $(f_i, \Delta_i), (f_j, \Delta_j)$ **do**
  if $(\Delta_i \leq \Delta_j) \wedge (f_i \geq f_j)$ **then**

remove (f_j, Δ_j)

After the list $L(2n-1)$ has been computed (there are $2n-1$ nodes in the graph formed by joining the two task graphs), $\alpha_d(\Delta)$ can be computed from it as follows:

$$\alpha_d(\Delta) = \max\{f \mid (f, \Delta') \in L(2n-1) \ \wedge \ \Delta' \leq \Delta\}$$

Finally, we state without proof the following theorem:

Theorem 1 *Given a task graph of n vertices, $\alpha_d(\Delta)$ can be computed in $O(n^3)$ time if the execution times of the subtasks are equal, and in general it can be computed in pseudo-polynomial time.*

3. STATIC PRIORITY SCHEDULING

The final section shortly describes the application of the above definitions and propositions to the schedulability test and priority assignment. To this end, we suppose that there are several tasks $T_1, ...T_i, ...T_n$, each one characterized by its request and demand curves α_r^i and α_d^i, respectively. They are all processed by one processing unit with delivery curve β using a static priority scheduler with preemption.

The following proposition allows to test whether a task with a given priority can be successfully scheduled on the processing unit. To this end, let us suppose that the tasks are ordered with decreasing priority, i.e. T_i has smaller priority than T_j if $i > j$.

Proposition 6 (Schedulability Test) *The capacity offered to a task T_i can be described by the remaining delivery curve β' according to Prop. 4 where $\alpha_r = \sum_{j=1}^{i-1} \alpha_r^i$. Task T_i meets all its deadlines if*

$$\beta'(\Delta) \geq \alpha_d^i(\Delta) \quad \forall \Delta$$

The proof can be given by using the concepts described in [15] and [16]. It has been shown in [15] that the above schedulability test reduces to the well known test in [14, 13, 9] if the respective task models are used.

Finally, we shortly describe how to determine priorities of the tasks such that the whole task system can be scheduled successfully, see [1]. To this end, we use the set of tasks $T_1, ..., T_n$. We select one task after the other while checking whether it can be scheduled. To this end, we assign the lowest priority to this task and apply Prop. 6. If such a task does not exist, the whole set can not be scheduled. Otherwise, the schedulable task is removed from the set $\mathcal{T}$ and the whole selection procedure is repeated until there is no task left. The sequence in which the tasks are removed corresponds to their increasing priority.

4. CONCLUDING REMARKS

The concepts described in this paper can be applied to other hard real-time scheduling algorithms such as EDF (earliest deadline first) as well. In addition, the request curves can be computed efficiently for other task models in a similar way. One of the major challenges is to combine the network calculus with the proposed real-time calculus in order to deal with distributed real-time systems within one homogeneous mathematical framework.

5. REFERENCES

[1] N. Audsley, A. Burns, M. Richardson, and A. Wellings, "Fixed priority preemptive scheduling: A historical perspective," *Real-Time Systems*, vol. 8, pp. 173–198, 1995.

[2] F. Baccelli, G. Cohen, G. Olsder, and J.-P. Quadrat, *Synchronization and Linearity*. New York: John Wiley, Sons, 1992.

[3] R. Cunninghame-Greene, "Minimax algebra," in *Lecture Note 166 in Economics and Mathematical Systems*, New York: Springer-Verlag, 1979.

[4] U. Schwiegelshohn and L. Thiele, "Dynamic min-max problems," *Discrete Event Dynamic Systems*, vol. 9, no. 2, pp. 111–134, 1999.

[5] J. L. Boudec, "Application of network calculus to guaranteed service networks," *IEEE Trans on Information theory*, vol. 44, May 1998.

[6] R. Cruz, "A calculus for network delay," *IEEE Trans. Information Theory*, vol. 37, no. 1, pp. 114–141, 1991.

[7] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks," *IEEE/ACM Trans. Networking*, vol. 2-2, pp. 137–150, 1994.

[8] R. Agrawal, R. L. Cruz, C. Okino, and R. Rajan, "Performance bounds for flow control protocols," *IEEE Transaction on Networks*, vol. 7, no. 3, pp. 310–323, 1999.

[9] A. Mok and D. Chen, "A multiframe model for real-time tasks," *IEEE Transactions on Software Engineering*, vol. 23, no. 10, pp. 635–645, 1997.

[10] S. Baruah, *A general model for recurringreal-time tasks*, pp. 114–122. Madrid, Spain: IEEE Computer Society Press, December 1998.

[11] K. Gresser, *"Echtzeitnachweis ereignisgesteuerter Realzeitsysteme"*. No. 268 in Fortschrittsberichte 10, VDI Verlag, 1993.

[12] M. Naedele, L. Thiele, and M. Eisenring, "Characterizing variable task releases and processor capacities," in *Proceedings of the 14th IFAC World Congress*, (Peking, China), pp. Q251–Q256, 1999.

[13] J. Lehoczky, L. Sha, and Y. Ding, *The rate monotonic scheduling algorithm*, pp. 166–171. Santa Monica, California, USA: IEEE Computer Society Press, December 1989.

[14] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[15] M. Naedele, L. Thiele, and M. Eisenring, "General task and resource models for processor task scheduling," 1998. Swiss Federal Institute of Technology, Computer Engineering and Networks Laboratory Report 45.

[16] S. Baruah, "Static-priority scheduling of recurring real-time tasks," 1999. Manuscript.